Project Manual

Topic - Walmart Store Sales Forecasting

TEAM MEMBERS-

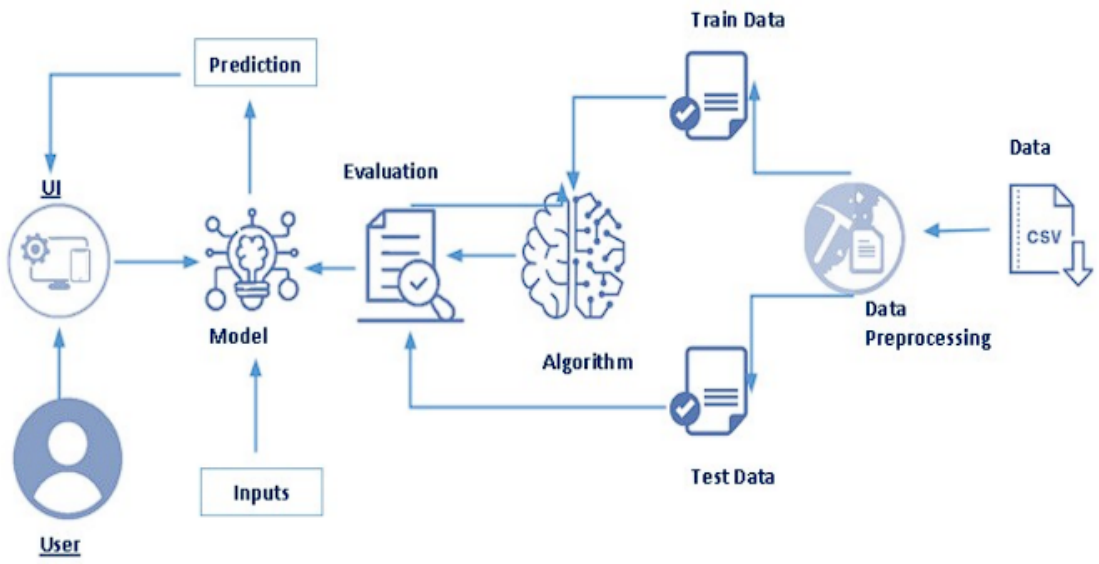
21BAI1677 - Bhardiwala Mitkumar Rajesh
21BAI1607- Aryan Tambekar
21BAI1598 - Abinav Shajil
21BAI1594 - Adamya Pundir

STEPS-

Project Description

Sales forecasting is the process of estimating future sales. Accurate sales forecasts enable companies to make informed business decisions and predict short-term and long-term performance. Companies can base their forecasts on past sales data, industry-wide comparisons, and economic trends. Here, the company is Walmart. Walmart is a renowned retail corporation that operates a chain of hypermarkets. Walmart has provided a data by combining the data of 45 stores including store information and monthly sales. Walmart runs several promotional markdown events throughout the year. These markdowns precede prominent holidays, the four largest of which are the Super Bowl, Labor Day, Thanksgiving, and Christmas. The weeks including these holidays are weighted five times higher in the evaluation than non-holiday weeks. The data is provided on weekly basis. We have to find the impact of holidays on the sales of the store. The holidays included are Christmas, Thanksgiving, Super Bowl and Labour Day. We will be using algorithms such as ARIMA, Random Forest, and XgBoost. We will train and test the data with these algorithms. Flask integration and IBM deployment will also be done.

ARCHITECTURE



TECHNOLOGIES USED

ANACONDA NAVIGATOR
HTML,CSS
FLASK

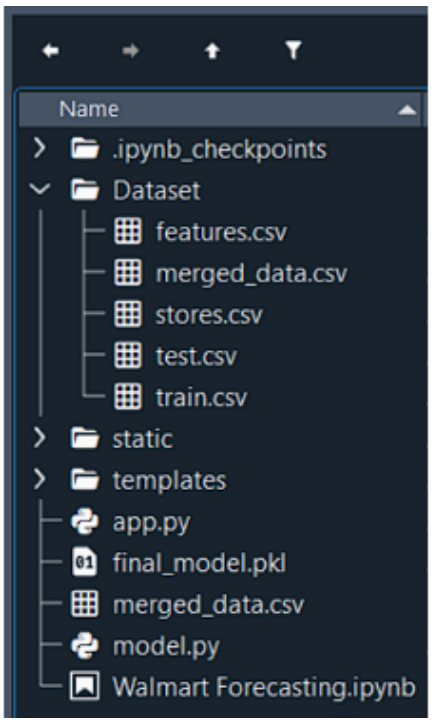
Project Flow

- The data is loaded and cleaning and feature extraction is performed on the data.
- Model is trained using machine learning models.
- Once model is trained, it is tested and evaluation is performed.

- Data collection
    - o Download the dataset from Kaggle
- Visualizing and analyzing data
    - o Descriptive analysis
- Data pre-processing
    - o Checking for null values
    - o Filling null values
    - o Splitting data into train and test
- Model building
    - o Import the model building libraries
    - o Initializing the model
    - o Training and testing the model
    - o Evaluating performance of model
    - o Save the model

Project Structure

Create the Project folder which contains files as shown below



We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.

- o Final.model.pkl is our saved model. This model will be used for flask integration.
- o Dataset folder contains model csv files – features, train, stores, test and merged_data .

o train.csv

- This is the historical training data, which covers to 2010–02–05 to

2012–11–01. The following fields are present:

- Store — the store number
- Dept — the department number
- Date — the week
- Weekly_Sales — sales for the given department in the given store
- IsHoliday — whether the week is a special holiday week

o test.csv

- This file is identical to train.csv, except the weekly sales column is omitted.
- (Note: This dataset will not be used in this project. Rather we will merge all the other datasets and use the merged data file for training and testing both.)

o features.csv

- This file contains additional data related to the store, department, and regional activity for the given dates. It contains the following fields:
- Store — the store number
- Date — the week
- Temperature — average temperature in the region
- Fuel_Price — cost of fuel in the region
- MarkDown1–5 — anonymized data related to promotional markdowns that Walmart is running. MarkDown data is only available after Nov 2011, and is not available for all stores all the time. Any missing value is marked with an NA.
- CPI — the consumer price index
- Unemployment — the unemployment rate
- IsHoliday — whether the week is a special holiday week
- The holidays considered are:
- Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12, 8-Feb-13
- Labor Day: 10-Sep-10, 9-Sep-11, 7-Sep-12, 6-Sep-13
- Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13
- Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13

Importing the libraries

The code imports essential libraries for data manipulation, statistical analysis, and machine learning, including NumPy, Pandas, SciPy Stats, Matplotlib, Seaborn, and Scikit-Learn. It also includes modules for evaluating regression models and handling date/time information using datetime and math libraries.

Importing the libraries

In [1]:
```
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
from datetime import datetime
import math
```

Importing the Dataset

The code reads three CSV files—'train.csv', 'features.csv', and 'stores.csv'—into Pandas DataFrames named 'train', 'features', and 'stores', respectively. These DataFrames likely represent datasets for a machine learning project or data analysis, with 'train' potentially containing training data, 'features' containing additional features, and 'stores' containing information about different stores.

Importing the Dataset

In [3]:
```
train=pd.read_csv("/kaggle/input/walmart-forecasting/train.csv")
features=pd.read_csv("/kaggle/input/walmart-forecasting/features.csv")
stores=pd.read_csv("/kaggle/input/walmart-forecasting/stores.csv")
```

Descriptive analysis

The code utilizes the describe() function on three Pandas DataFrames—'train', 'features', and 'stores'—providing summary statistics such as mean, standard deviation, minimum, maximum, and quartile information for each numeric column in the datasets. This helps to quickly understand the distribution and characteristics of the data in each DataFrame.

Descriptive analysis

In [7]:
```
train.describe()
```

Out[7]:

	Store	Dept	Weekly_Sales
count	421570.000000	421570.000000	421570.000000
mean	22.200546	44.260317	15981.258123
std	12.785297	30.492054	22711.183519
min	1.000000	1.000000	-4988.940000
25%	11.000000	18.000000	2079.650000
50%	22.000000	37.000000	7612.030000
75%	33.000000	74.000000	20205.852500
max	45.000000	99.000000	693099.360000

In [8]:
```
features.describe()
```

Out[8]:

	Store	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment
count	8190.000000	8190.000000	8190.000000	4032.000000	2921.000000	3613.000000	3464.000000	4050.000000	7605.000000	7605.000000
mean	23.000000	59.356198	3.405992	7032.371786	3384.176594	1760.100180	3292.935886	4132.216422	172.460809	7.826821
std	12.987966	18.678607	0.431337	9262.747448	8793.583016	11276.462208	6792.329861	13086.690278	39.738346	1.877259
min	1.000000	-7.290000	2.472000	-2781.450000	-265.760000	-179.260000	0.220000	-185.170000	126.064000	3.684000
25%	12.000000	45.902500	3.041000	1577.532500	68.880000	6.600000	304.687500	1440.827500	132.364839	6.634000
50%	23.000000	60.710000	3.513000	4743.580000	364.570000	36.260000	1176.425000	2727.135000	182.764003	7.806000
75%	34.000000	73.880000	3.743000	8923.310000	2153.350000	163.150000	3310.007500	4832.555000	213.932412	8.567000
max	45.000000	101.950000	4.468000	103184.980000	104519.540000	149483.310000	67474.850000	771448.100000	228.976456	14.313000

In [9]:
```
stores.describe()
```

Out[9]:

	Store	Size
count	45.000000	45.000000
mean	23.000000	130287.600000
std	13.133926	63825.271991
min	1.000000	34875.000000
25%	12.000000	70713.000000
50%	23.000000	126512.000000
75%	34.000000	202307.000000
max	45.000000	219622.000000

Data Pre-processing

CHECKING FOR NULL VALUES

The code uses the info() method on three Pandas DataFrames—'train', 'features', and 'stores'—to display concise information about each dataset, including the data types of columns, non-null counts, and memory usage. This is useful for understanding the structure and completeness of the data in each DataFrame.

Data Pre-processing

Checking for null values

In [10]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 421570 entries, 0 to 421569
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Store         421570 non-null  int64
 1   Dept          421570 non-null  int64
 2   Date          421570 non-null  object
 3   Weekly_Sales  421570 non-null  float64
 4   IsHoliday     421570 non-null  bool
dtypes: bool(1), float64(1), int64(2), object(1)
memory usage: 13.3+ MB
```

In [11]:
```
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8190 entries, 0 to 8189
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         8190 non-null   int64
 1   Date          8190 non-null   object
 2   Temperature   8190 non-null   float64
 3   Fuel_Price    8190 non-null   float64
 4   MarkDown1     4032 non-null   float64
 5   MarkDown2     2921 non-null   float64
 6   MarkDown3     3613 non-null   float64
 7   MarkDown4     3464 non-null   float64
 8   MarkDown5     4050 non-null   float64
 9   CPI           7605 non-null   float64
 10  Unemployment  7605 non-null   float64
 11  IsHoliday     8190 non-null   bool
dtypes: bool(1), float64(9), int64(1), object(1)
memory usage: 712.0+ KB
```

In [12]:
```
stores.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45 entries, 0 to 44
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Store   45 non-null     int64
 1   Type    45 non-null     object
 2   Size    45 non-null     int64
dtypes: int64(2), object(1)
memory usage: 1.2+ KB
```

MERGING DATASET

The code performs two successive merges using Pandas' merge() function. First, it merges the 'train' and 'stores' DataFrames on the 'Store' column. Then, it merges the resulting DataFrame with the 'features' DataFrame on the columns 'Store', 'Date', and 'IsHoliday'. This likely combines information from multiple datasets based on store identifiers, date, and holiday status.

merging datasets

In [16]:
```
data = pd.merge(train, stores, on='Store')
data = pd.merge(data, features, on=['Store', 'Date','IsHoliday'])
```

In [17]:
```
data.head()
```

Out[17]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Type	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	
0	1	1	2010-02-05	24924.50	False	A	151315	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096
1	1	2	2010-02-05	50605.27	False	A	151315	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096
2	1	3	2010-02-05	13740.12	False	A	151315	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096
3	1	4	2010-02-05	39954.04	False	A	151315	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096
4	1	5	2010-02-05	32229.38	False	A	151315	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096

CONVERTING DATE TO RESPECTIVE DAYS,MONTH AND YEAR

The code manipulates the 'Date' column in the DataFrame 'data': it converts it to a datetime format, extracts 'Month,' 'Year,' and 'Day_of_Week,' and creates additional columns, 'Day_of_Week' with day names and 'Is_Weekend' indicating whether the date falls on a Saturday or Sunday. This preprocessing enhances temporal features, facilitating time-based analysis in the dataset.

Converting Date to respective Days,Month and Year

In [21]:
```
# Convert the 'Date' column to a datetime format
data['Date'] = pd.to_datetime(data['Date'])

# Extract 'Month,' 'Year,' and 'Day_of_Week' from the 'Date' column
data['Month'] = data['Date'].dt.month
data['Year'] = data['Date'].dt.year

# Create a 'Day_of_Week' column with day names
data['Day_of_Week'] = data['Date'].dt.strftime('%A')

# Create an 'Is_Weekend' column with True for Saturday and Sunday
data['Is_Weekend'] = (data['Day_of_Week'].isin(['Saturday', 'Sunday'])).astype(bool)
```

DATA ENCODING

The code defines mappings for categorical columns ('Is_Holiday', 'Type', 'Day_of_Week', 'Is_Weekend') by creating dictionaries. It then applies these mappings to corresponding columns in the 'data' DataFrame, converting categorical values into numerical representations. Finally, it prints the defined mappings for reference. This preprocessing step prepares the data for machine learning algorithms that require numerical inputs.

Data Encoding

In [24]:
```
# Define mappings for Is_Holiday
is_holiday_mapping = {False: 0, True: 1}

# Define mappings for Type
type_mapping = {'A': 0, 'B': 1, 'C': 2}

# Define mappings for Day_of_Week
day_of_week_mapping = {
    'Monday': 0, 'Tuesday': 1, 'Wednesday': 2, 'Thursday': 3, 'Friday': 4, 'Saturday': 5, 'Sunday': 6
}

# Define mappings for Is_Weekend
is_weekend_mapping = {False: 0, True: 1}

# Apply mappings to the dataset
data['IsHoliday'] = data['IsHoliday'].map(is_holiday_mapping)
data['Type'] = data['Type'].map(type_mapping)
data['Day_of_Week'] = data['Day_of_Week'].map(day_of_week_mapping)
data['Is_Weekend'] = data['Is_Weekend'].map(is_weekend_mapping)

# Print all mappings
print("Is_Holiday Mapping:")
print(is_holiday_mapping)
print("\nType Mapping:")
print(type_mapping)
print("\nDay_of_Week Mapping:")
print(day_of_week_mapping)
print("\nIs_Weekend Mapping:")
print(is_weekend_mapping)
```

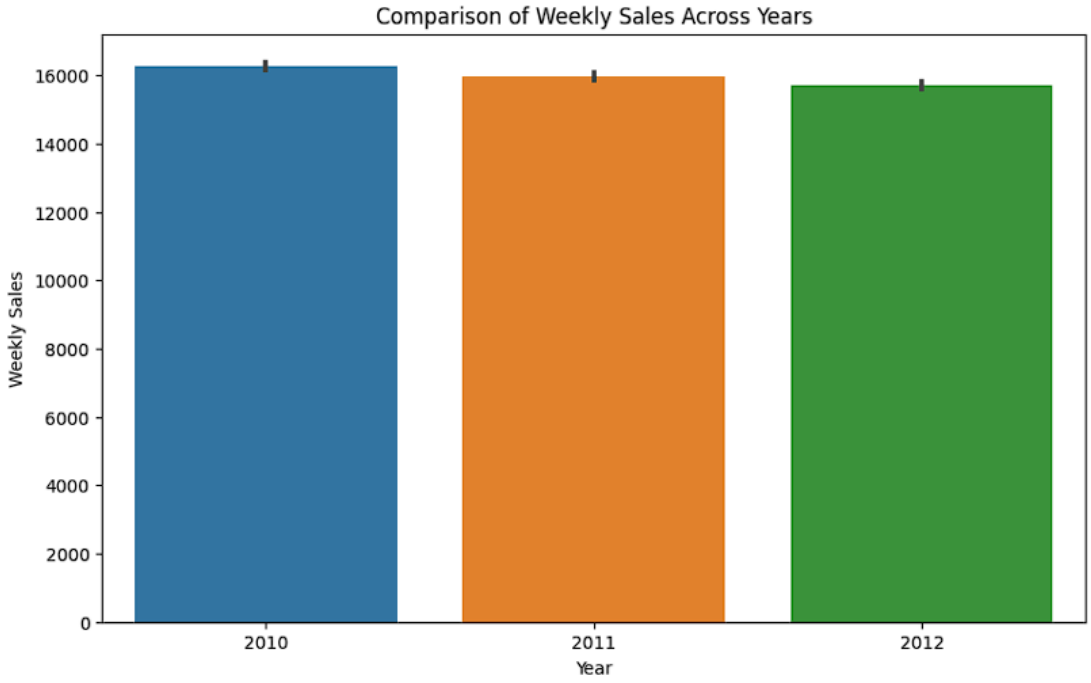
```
Is_Holiday Mapping:
{False: 0, True: 1}

Type Mapping:
{'A': 0, 'B': 1, 'C': 2}

Day_of_Week Mapping:
{'Monday': 0, 'Tuesday': 1, 'Wednesday': 2, 'Thursday': 3, 'Friday': 4, 'Saturday': 5, 'Sunday': 6}

Is_Weekend Mapping:
{False: 0, True: 1}
```

DATA VISUALIZATION

The code generates three bar plots using Seaborn and Matplotlib. The first compares 'Weekly_Sales' across different years, providing insights into sales trends over time. The second displays a bar plot of the distribution of years in the dataset, indicating the frequency of each year. The third visualizes the count of 'IsHoliday,' providing an overview of the occurrence of holidays in the dataset. These visualizations offer valuable insights into sales patterns, the distribution of years, and the presence of holidays in the dataset.

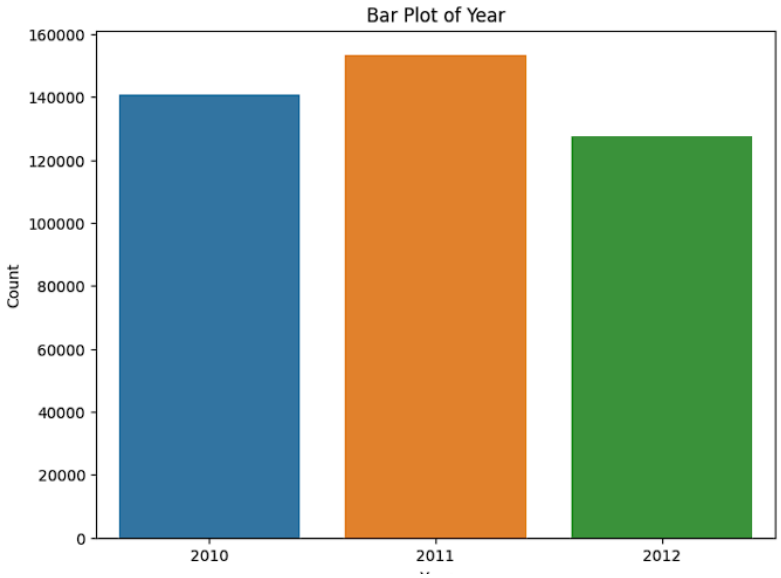
Data Visualization

In [38]:
```
# Create a bar plot for Weekly_Sales across different years
plt.figure(figsize=(10, 6))
sns.barplot(x='Year', y='Weekly_Sales', data=data)  # ci=None removes error bars

# Add labels and title
plt.xlabel('Year')
plt.ylabel('Weekly Sales')
plt.title('Comparison of Weekly Sales Across Years')

# Show the plot
plt.show()
```



It shows that all the years have almost equal value for weekly sales.

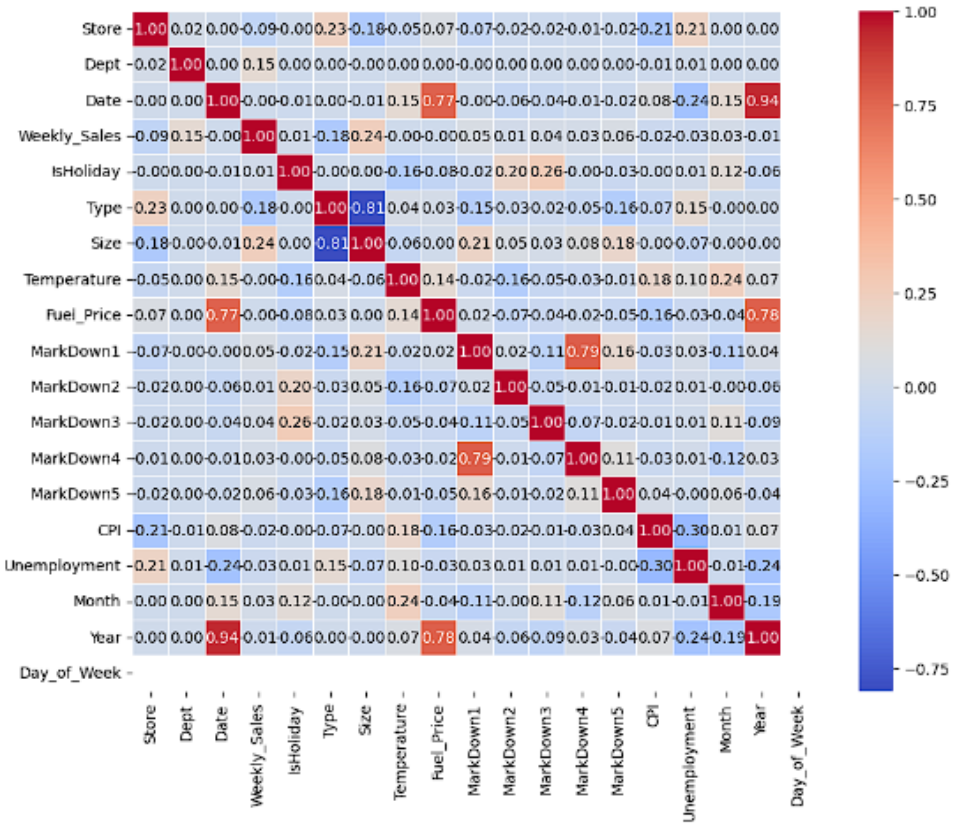
In [36]:
```
# Create a bar plot for the "Year" column
plt.figure(figsize=(8, 6))
sns.barplot(x=data['Year'].value_counts().index, y=data['Year'].value_counts())

# Add labels and title
plt.xlabel('Year')
plt.ylabel('Count')
plt.title('Bar Plot of Year')

# Show the plot
plt.show()
```



DATA CORRELATION

The code calculates the correlation matrix for numeric columns in the 'data' DataFrame, representing the relationships between variables. The resulting matrix is visualized as a heatmap using Seaborn and Matplotlib, with correlation values annotated for clarity. This heatmap helps identify patterns and dependencies among different features, offering insights into potential relationships within the dataset. The 'coolwarm' color map is employed, where warmer colors indicate stronger positive correlations, and cooler colors indicate stronger negative correlations.

In [32]:
```
# Compute the correlation matrix
corr = data.corr()

# Increase the height and width of the heatmap
plt.figure(figsize=(10, 8))

# Visualize the correlation matrix using a heatmap
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5)

# Show the plot
plt.show()
```



MODEL TRAINING AND TESTING

We trained our model using 3 different techniques-

1)RANDOM FOREST

The code implements a Random Forest Regressor using scikit-learn for predicting 'Weekly_Sales' based on the features in the dataset. It splits the data into training and testing sets, initializes the Random Forest model with 100 trees, and evaluates its performance on the test set. The evaluation metrics include R-squared (accuracy), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE), providing a comprehensive assessment of the model's predictive capabilities. The results are then printed for further analysis and interpretation.

1)Random forest

In [44]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
import numpy as np

# Define your features (X) and the target variable (y)
X_rf = data.drop(columns=['Weekly_Sales','Date'])
y_rf = data['Weekly_Sales']

# Split the data into a training set (70%) and a testing set (30%)
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X_rf, y_rf, test_size=0.3, random_state=42)

# Initialize and train the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train_rf, y_train_rf)

# Make predictions on the test data
y_pred_rf = rf_model.predict(X_test_rf)

# Calculate R-squared (accuracy), RMSE, and MAE
r2_rf = r2_score(y_test_rf, y_pred_rf)
rmse_rf = np.sqrt(mean_squared_error(y_test_rf, y_pred_rf))
mae_rf = mean_absolute_error(y_test_rf, y_pred_rf)

print("Random Forest Model:")
print("R-squared (Accuracy):", r2_rf)
print("Root Mean Squared Error (RMSE):", rmse_rf)
print("Mean Absolute Error (MAE):", mae_rf)
```

```
Random Forest Model:
R-squared (Accuracy): 0.9625817498533463
Root Mean Squared Error (RMSE): 4438.228933960859
Mean Absolute Error (MAE): 1648.1275767946804
```

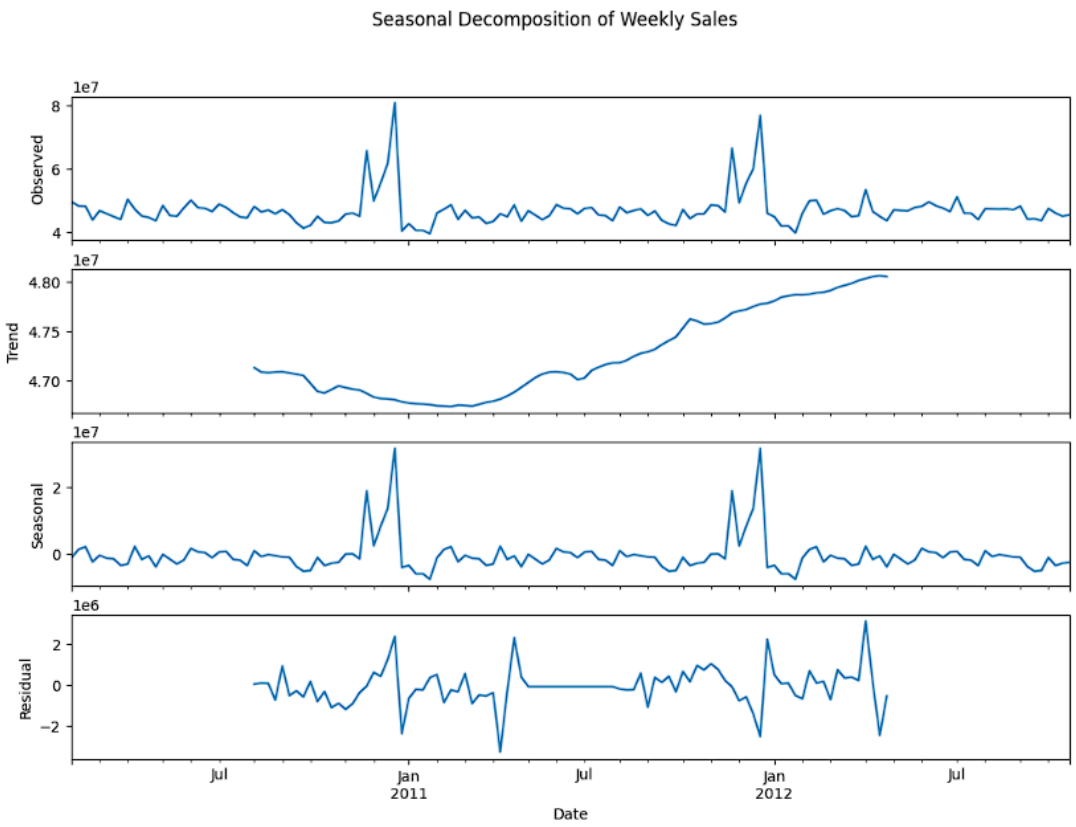
2)ARIMA MODEL

The code conducts time series analysis using the auto ARIMA model for 'Weekly_Sales.' It preprocesses the data, aggregates sales by date, and splits it into training and testing sets. The auto ARIMA model is trained, and predictions are evaluated using R-squared, Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE). Additionally, the code provides a seasonal decomposition plot of the observed data into trend, seasonal, and residual components, offering insights into the underlying patterns. The printed evaluation metrics summarize the model's performance in forecasting weekly sales.

2) ARIMA model

In [42]:
```
import pandas as pd
import numpy as np
from pmdarima import auto_arima
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Assuming 'data' is your DataFrame
# If 'Date' is not in datetime format, convert it
data['Date'] = pd.to_datetime(data['Date'])

# Selecting relevant columns for the time series analysis
time_series_columns = ['Date', 'Weekly_Sales']
df_time_series = data[time_series_columns]

# Aggregating sales by date
df_time_series = df_time_series.groupby('Date')['Weekly_Sales'].sum().reset_index()

# Splitting the dataset into training and testing sets
train_size = int(len(df_time_series) * 0.8)
train, test = df_time_series[:train_size], df_time_series[train_size:]

# Performing auto ARIMA model training
model = auto_arima(train['Weekly_Sales'], suppress_warnings=True, seasonal=True, stepwise=True)
forecast, conf_int = model.predict(n_periods=len(test), return_conf_int=True)

# Evaluating the model
r2_arima = r2_score(test['Weekly_Sales'], forecast)
rmse_arima = np.sqrt(mean_squared_error(test['Weekly_Sales'], forecast))
mae_arima = mean_absolute_error(test['Weekly_Sales'], forecast)
```

```
# Printing the evaluation metrics
print("Auto ARIMA Model:")
print("R-squared (Accuracy):", r2_arima)
print("Root Mean Squared Error (RMSE):", rmse_arima)
print("Mean Absolute Error (MAE):", mae_arima)
```



3)XG BOOST

The code employs XGBoost, a gradient boosting algorithm, to predict 'Weekly_Sales' based on features such as 'Size,' 'Temperature,' 'Fuel_Price,' 'MarkDown1-5,' 'CPI,' 'Unemployment,' 'Month,' 'Year,' and 'Day_of_Week.' The dataset is split into training and testing sets, and the XGBoost model is trained and evaluated. Evaluation metrics, including R-squared (accuracy), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE), are calculated and printed. This summary provides an overview of the XGBoost model's performance in predicting weekly sales.

3)XG Boost

Type *Markdown* and LaTeX: α^2

In [43]:
```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split

# Load your dataset
# Replace 'your_dataset.csv' with the actual file path or URL of your dataset

# Define your features (X) and target (y)
# In this case, we will use 'Size', 'Temperature', 'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5',
# Define your features (X) and the target variable (y)
X= data.drop(columns=['Weekly_Sales','Date'])
y = data['Weekly_Sales']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train the XGBoost model
xg_model = xgb.XGBRegressor()
xg_model.fit(X_train, y_train)

# Make predictions on the test data
predictions = xg_model.predict(X_test)

# Calculate and print R-squared (accuracy), RMSE, and MAE
r2_xgboost = r2_score(y_test, predictions)
rmse_xgboost = np.sqrt(mean_squared_error(y_test, predictions))
mae_xgboost = mean_absolute_error(y_test, predictions)

print("XGBoost Model:")
print("R-squared (Accuracy):", r2_xgboost)
print("Root Mean Squared Error (RMSE):", rmse_xgboost)
print("Mean Absolute Error (MAE):", mae_xgboost)
```

```
XGBoost Model:
R-squared (Accuracy): 0.9331301092436165
Root Mean Squared Error (RMSE): 5933.121885368714
Mean Absolute Error (MAE): 3180.4101097696407
```

According to the results, random forest performed the best with accuracy of around 96%. Then we saved model.pkl file with random forest model.

Saving Model.pkl file

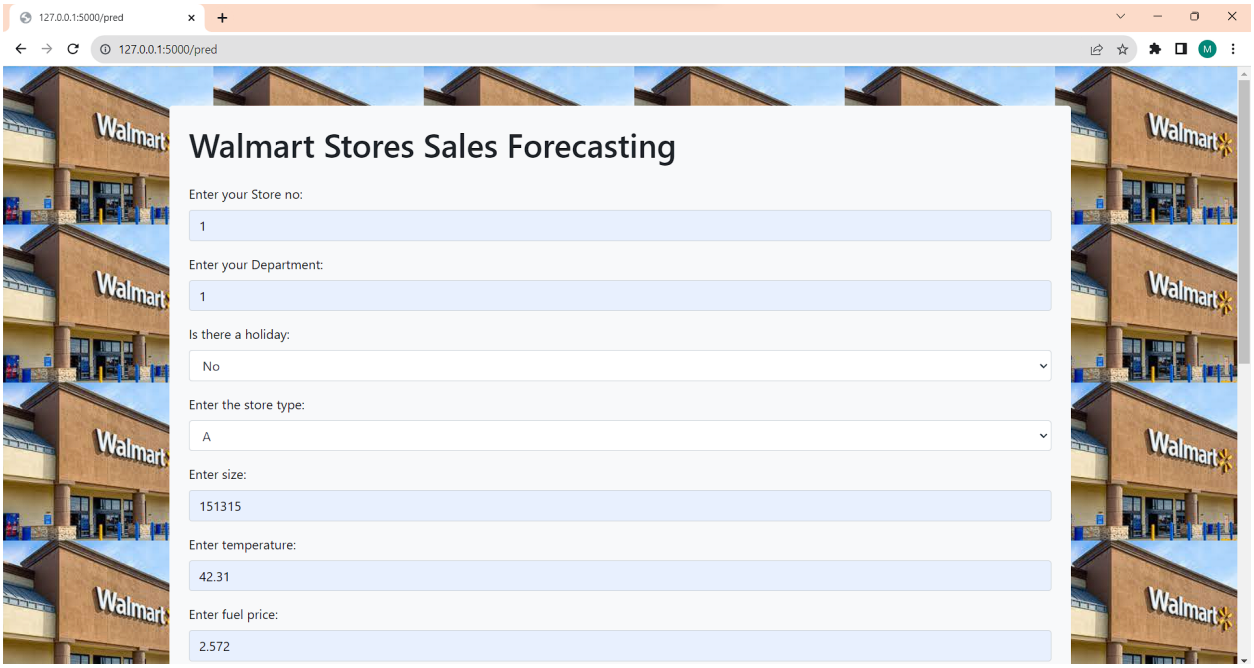
In [ ]:
```
import pickle
pickle.dump(rf_model,open('model.pkl','wb'))
```

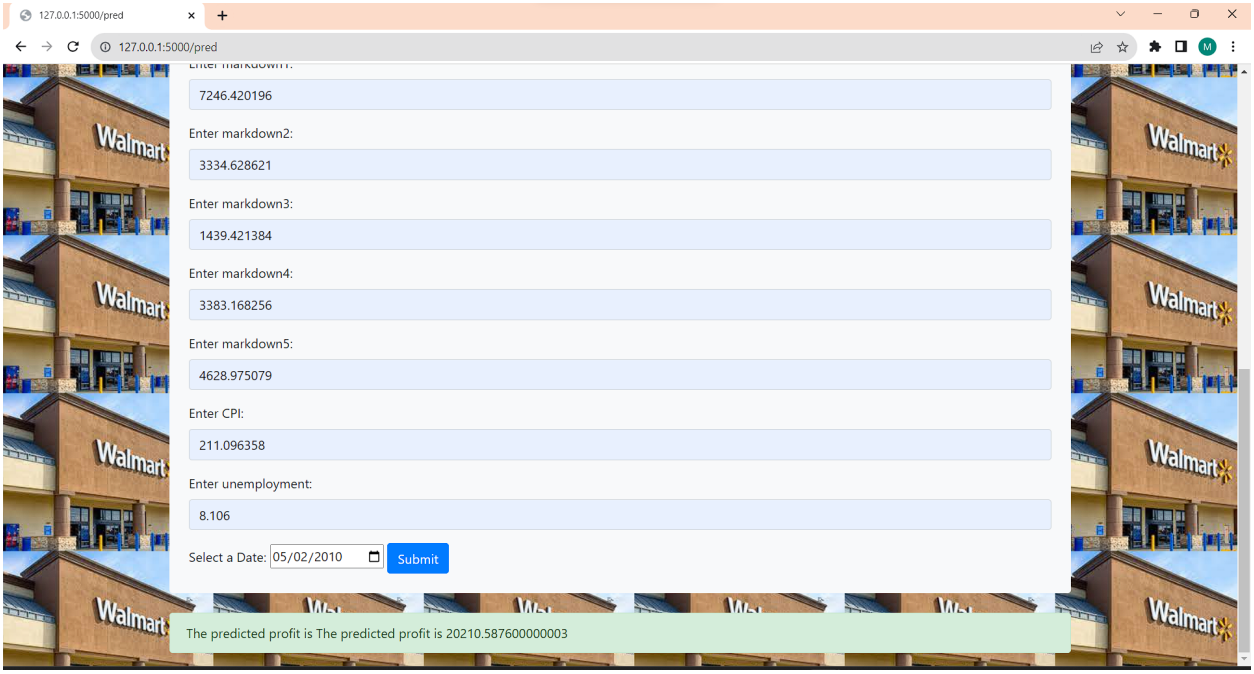
In my Flask application, I've organized my static resources, including images, in a 'static' folder. The HTML templates, specifically 'home.html,' are stored in a 'templates' folder. The HTML file incorporates Bootstrap for styling and features a form for Walmart Stores Sales Forecasting.

The Flask application ('app.py') uses the 'model.pkl' file, created in a Jupyter notebook, to predict profits based on user inputs. The web form collects data such as store number, department, holiday status, store type, and various other features. The predicted profit is displayed in a separate container with a success message.

The application is styled with a background image and Bootstrap components. The 'date' input is processed to extract relevant date components. Overall, the application is designed for predicting profits using a pre-trained model, offering a user-friendly interface for input and result display.

The final output looks like this-




127.0.0.1:5000/pred
127.0.0.1:5000/pred
7246.420196
Enter markdown2:
3334.628621
Enter markdown3:
1439.421384
Enter markdown4:
3383.168256
Enter markdown5:
4628.975079
Enter CPI:
211.096358
Enter unemployment:
8.106
Select a Date: 05/02/2010
Submit
The predicted profit is The predicted profit is 20210.587600000003

****